CMSC416 Instructor Notes Spring 2024
Topics: Message Passing and MPI

1 Shared Memory vs. Distributed Memory Systems

Shared Memory Systems: In these systems, all processors have access to a common memory space. This can be uniform (UMA) where each processor has equal access time to memory or non-uniform (NUMA) where access time varies depending on memory location.

Distributed Memory Systems: Here, each processor has its own private memory. Processes communicate by sending and receiving messages. This model is the foundation of MPI.

Shared Memory Models: Utilize threads where all threads have access to the entire memory space. Examples include Pthreads and OpenMP.

Distributed Memory Models (Message Passing Models): Utilize processes with each having access to its own memory space. MPI is a prime example of this model.

Hybrid Models: Combine shared and distributed memory models, often using OpenMP within nodes and MPI across nodes for efficiency.

2 Basic MPI

Message Passing Interface (MPI): A standard for writing parallel programs. MPI allows for communication between processes by passing messages. It's essential for distributed computing environments where processes may run on different physical machines. MPI programs follow the SPMD (Single Program Multiple Data) model, where all processes execute the same program but may perform different operations based on their rank. Main MPI routines:

- MPI_Init and MPI_Finalize: Required to start and end any MPI program, initializing and cleaning up the MPI environment.
- MPI_Comm_size and MPI_Comm_rank: Used to determine the size of the communicator (total number of processes) and the rank (the process ID) within that communicator.
- MPI_Send and MPI_Recv: Core functions for sending and receiving messages between processes.

Important Points:

- MPI is designed for portability across various hardware architectures.
- Proper initialization and finalization of MPI programs are crucial for resource management and avoiding runtime issues.
- MPI uses ranks to identify processes. These are integer IDs assigned to each process within a communicator.
- Communicators in MPI define a group of processes that can communicate with each other. `MPI_COMM_WORLD` is a default communicator that includes all processes.

To compile an MPI program, one uses 'mpicc' followed by the source code filename. Running an MPI program involves specifying the number of processes with 'mpiexec' or 'mpirun'. Proper allocation of resources (e.g., number of processors) is essential when running MPI programs to avoid inefficiencies or runtime errors.

2.1 Blocking Communication in MPI

MPI provides mechanisms for sending and receiving messages between processes. The basic operations for this are 'MPI_Send' and 'MPI_Recv'. These operations are blocking, meaning the 'MPI_Send' operation will not complete until the message has been copied out of the send buffer, and 'MPI_Recv' will not complete until the received message has been copied into the receive buffer. Deadlocks can occur if two processes wait on each other to receive messages. Careful programming is required to avoid these situations.

3 Non-Blocking Communication

Involves operations that allow a program to initiate a communication request and proceed without waiting for the operation to complete. This is achieved using request objects that link a non-blocking operation with its completion.

Request Objects: Opaque objects managed by MPI to track non-blocking operations. They are used with functions like MPI Wait or MPI Test to check for the completion of non-blocking sends and receives.

MPI_Isend and MPI_Irecv receive allow processes to perform other computations while waiting for communication to complete. This is useful for overlapping computation with communication, potentially improving overall performance.

4 Communicators

A communicator defines a group of processes that can communicate with one another. Processes within a communicator are assigned unique ranks. This system allows for flexible and organized communication patterns among processes.

Creating Sub-communicators: Useful for organizing processes into smaller groups for specific tasks, such as operations within rows or columns of a virtual proccess grid.

5 Collective Operations

Operations that involve all processes within a communicator. Examples include MPI Barrier, MPI Bcast, MPI Reduce, MPI Allreduce, MPI Scatter, and MPI Gather. These operations facilitate synchronization, data distribution, aggregation, and collection among processes.

MPI_Barrier: A synchronization operation that blocks processes at a certain point until all processes in the communicator reach the barrier.

MPI_Bcast: Broadcasts data from one process (the root) to all other processes in a communicator. This is a collective operation that simplifies data distribution tasks. It demonstrates how MPI treats buffers differently based on whether a process is the root or a receiver.

MPI_Reduce: Aggregates data from all processes in a communicator to a single process, using an operation like sum, min, or max. It requires separate send and receive buffers, with the receive buffer only needed at the root process.

MPI Allreduce: Similar to MPI Reduce, but the zesult of the aggregation is distributed back to all processes.1

`MPI_Scatter` and `MPI_Gather`: Scatter is used to distribute distinct pieces of data from a root process to all processes in a communicator, while Gather collects distinct pieces of data from all processes to a root process. Variants like Scatterv and Gatherv allow for different amounts of data to be sent or received.

MPI Wtime: A function for measuring elapsed time, useful for benchmarking parts of a program.

6 Message Passing Protocols in MPI

MPI uses different protocols for sending messages, depending on the size.

Eager Protocol: Used for small messages. The MPI runtime sends the message without waiting for the receiver to allocate buffer space.

Rendezvous Protocol: Used for larger messages. It involves a handshake between the sender and receiver to ensure buffer space is allocated before sending the message.

Short Message Protocol: Similar to eager but for very short messages, where data is sent in the message envelope itself.

The MPI runtime automatically decides which protocol to use based on message size. However, you can adjust the threshold sizes using environment variables to optimize performance.

7 MPI and Network Communication

MPI abstracts the details of the underlying network and hardware, ensuring portability. It interacts with lower-level communication libraries provided by hardware vendors, like UCX or PSM.

When sending a message, it goes from the process's memory to the network interface card (NIC), through potentially multiple network switches (with their buffers), to the destination NIC, and finally to the receiving process's memory.